Joey NMT: A Minimalist NMT Toolkit for Novices

Julia Kreutzer
Computational Linguistics
Heidelberg University
kreutzer@cl.uni-heidelberg.de

Jasmijn Bastings
ILLC
University of Amsterdam
bastings@uva.nl

Stefan Riezler
Computational Linguistics & IWR
Heidelberg University
riezler@cl.uni-heidelberg.de

Abstract

We present Joey NMT, a minimalist neural machine translation toolkit based on PyTorch that is specifically designed for novices. Joey NMT provides many popular NMT features in a small and simple code base, so that novices can easily and quickly learn to use it and adapt it to their needs. Despite its focus on simplicity, Joey NMT supports classic architectures (RNNs, transformers), fast beam search, weight tying, and more, and achieves performance comparable to more complex toolkits on standard benchmarks. We evaluate the accessibility of our toolkit in a user study where novices with general knowledge about Pytorch and NMT and experts work through a self-contained Joey NMT tutorial, showing that novices perform almost as well as experts in a subsequent code quiz. Joey NMT is available at https://github.com/joeynmt/joeynmt.

1 Introduction

Since the first successes of neural machine translation (NMT), various research groups and industry labs have developed open source toolkits specialized for NMT, based on new open source deep learning platforms. While toolkits like OpenNMT (Klein et al., 2018), XNMT (Neubig et al., 2018) and Neural Monkey (Helcl and Libovický, 2017) aim at readability and extensibility of their codebase, their target group are researchers with a solid background in machine translation and deep learning, and with experience in navigating, understanding and handling large code bases. However, none of the existing NMT tools has been designed primarily for readability or accessibility for novices, nor has anyone studied quality and accessibility of such code empirically. On the other hand, it is an important challenge for novices to understand how NMT is implemented, what features each toolkit implements exactly, and which toolkit to choose in order to code their own project as fast and simple as possible.

We present an NMT toolkit especially designed for novices, providing clean, well documented, and minimalistic code, that is yet of comparable quality to more complex codebases on standard benchmarks. Our approach is to identify the core features of NMT that have not changed over the last years, and to invest in documentation, simplicity and quality of the code. These core features include standard network architectures (RNN, transformer, different attention mechanisms, input feeding, configurable encoder/decoder bridge), standard learning techniques (dropout, learning rate scheduling, weight tying, early stopping criteria), and visualization/monitoring tools.

We evaluate our codebase in several ways: Firstly, we show that Joey NMT's comment-to-code ratio is almost twice as high as other toolkits which are roughly 9-10 times larger. Secondly, we present an evaluation on standard benchmarks (WMT17, IWSLT) where we show that the core architectures implemented in Joey NMT achieve comparable performance to more complex state-of-the-art toolkits. Lastly, we conduct a user study where we test the code understanding of novices, i.e. students with basic knowledge about NMT and PyTorch, against expert coders. While novices, after having worked through a self-contained Joey NMT tutorial, needed more time to answer each question in an in-depth code quiz, they achieved only marginally lower scores than the experts. To our knowledge, this is the first user study on the accessibility of NMT toolkits.

2 Joey NMT

2.1 NMT Architectures

This section formalizes the Joey NMT implementation of autoregressive recurrent and fully-

Proceedings of the 2019 EMNLP and the 9th IJCNLP (System Demonstrations), pages 109–114
Hong Kong, China, November 3 – 7, 2019. ©2019 Association for Computational Linguistics

attentional models.

In the following, a source sentence of length l_x is represented by a sequence of one-hot encoded vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{l_x}$ for each word. Analogously, a target sequence of length l_y is represented by a sequence of one-hot encoded vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{l_y}$.

2.1.1 RNN

Joey NMT implements the RNN encoder-decoder variant from Luong et al. (2015).

Encoder. The encoder RNN transforms the input sequence $\mathbf{x}_1, \ldots, \mathbf{x}_{l_x}$ into a sequence of vectors $\mathbf{h}_1, \ldots, \mathbf{h}_{l_x}$ with the help of the embeddings matrix E_{src} and a recurrent computation of states

$$\mathbf{h}_i = \text{RNN}(E_{src}\,\mathbf{x}_i, \mathbf{h}_{i-1}); \qquad \mathbf{h}_0 = \mathbf{0}.$$

The RNN consists of either GRU or a LSTM units. For a bidirectional RNN, hidden states from both directions are are concatenated to form $\mathbf{h}_i$. The initial encoder hidden state $\mathbf{h}_0$ is a vector of zeros. Multiple layers can be stacked by using each resulting output sequence $\mathbf{h}_1, \ldots, \mathbf{h}_{l_x}$ as the input to the next RNN layer.

Decoder. The decoder uses input feeding (Luong et al., 2015) where an attentional vector $\tilde{\mathbf{s}}$ is concatenated with the representation of the previous word as input to the RNN. Decoder states are computed as follows:

$$\begin{aligned}
\mathbf{s}_t &= \text{RNN}([E_{trg}\,\mathbf{y}_{t-1}; \tilde{\mathbf{s}}_{t-1}], \mathbf{s}_{t-1}) \\
\mathbf{s}_0 &= \begin{cases} \tanh(W_{bridge}\,\mathbf{h}_{l_x} + \mathbf{b}_{bridge}) & \text{if bridge} \\ \mathbf{h}_{l_x} & \text{if last} \\ \mathbf{0} & \text{otherwise} \end{cases} \\
\tilde{\mathbf{s}}_t &= \tanh(W_{att}[\mathbf{s}_t; \mathbf{c}_t] + \mathbf{b}_{att})
\end{aligned}$$

The initial decoder state is configurable to be either a non-linear transformation of the last encoder state (“bridge”), or identical to the last encoder state (“last”), or a vector of zeros.

Attention. The context vector $\mathbf{c}_t$ is computed with an attention mechanism scoring the previous decoder state $\mathbf{s}_{t-1}$ and each encoder state $\mathbf{h}_i$:

$$\begin{aligned}
\mathbf{c}_t &= \sum_i a_{ti} \cdot \mathbf{h}_i \\
a_{ti} &= \frac{\exp(\text{score}(\mathbf{s}_{t-1}, \mathbf{h}_i))}{\sum_k \exp(\text{score}(\mathbf{s}_{t-1}, \mathbf{h}_k))}
\end{aligned}$$

where the scoring function is a multi-layer perceptron (Bahdanau et al., 2015) or a bilinear transformation (Luong et al., 2015).

Output. The output layer produces a vector $\mathbf{o}_t = W_{out}\,\tilde{\mathbf{s}}_t$, which contains a score for each token in the target vocabulary. Through a softmax transformation, these scores can be interpreted as a probability distribution over the target vocabulary $\mathcal{V}$ that defines an index over target tokens v_j.

$$p(y_t = v_j \mid x, y_{<t}) = \frac{\exp(\mathbf{o}_t[j])}{\sum_{k=1}^{|\mathcal{V}|} \exp(\mathbf{o}_t[k])}$$

2.1.2 Transformer

Joey NMT implements the Transformer from Vaswani et al. (2017), with code based on *The Annotated Transformer* blog (Rush, 2018).

Encoder. Given an input sequence $\mathbf{x}_1, \ldots, \mathbf{x}_{l_x}$, we look up the word embedding for each input word using $E_{src}\mathbf{x}_i$, add a position encoding to it, and stack the resulting sequence of word embeddings to form matrix $X \in \mathbb{R}^{l_x \times d}$, where l_x is the sentence length and d the dimensionality of the embeddings.

We define the following learnable parameters:[1]

$$A \in \mathbb{R}^{d \times d_a} \quad B \in \mathbb{R}^{d \times d_a} \quad C \in \mathbb{R}^{d \times d_o}$$

where d_a is the dimensionality of the attention (inner product) space and d_o the output dimensionality. Transforming the input matrix with these matrices into new word representations H

$$H = \underbrace{\text{softmax}\big(XA\,B^\top X^\top\big)}_{\text{self-attention}}\ XC$$

which have been updated by attending to all other source words. Joey NMT implements multi-headed attention, where this transformation is computed k times, one time for each head with different parameters A, B, C.

After computing all k Hs in parallel, we concatenate them and apply layer normalization and a final feed-forward layer:

$$\begin{aligned}
H &= [H^{(1)}; \ldots; H^{(k)}] \\
H' &= \text{layer-norm}(H) + X \\
H^{(\text{enc})} &= \text{feed-forward}(H') + H'
\end{aligned}$$

We set $d_o = d/k$, so that $H \in \mathbb{R}^{l_x \times d}$. Multiple of these layers can be stacked by setting $X = H^{(\text{enc})}$ and repeating the computation.

[1] Exposition adapted from Michael Collins https://youtu.be/jfwqRMdTmLo

Decoder. The Transformer decoder operates in a similar way as the encoder, but takes the stacked target embeddings $Y \in \mathbb{R}^{l_y \times d}$ as input:

$$H = \underbrace{\text{softmax}\big(YA\,B^{\top}Y^{\top}\big)}_{\text{masked self-attention}}\,YC$$

For each target position attention to future input words is inhibited by setting those attention scores to $-inf$ before the softmax. After obtaining $H' = H + Y$, and before the feed-forward layer, we compute multi-headed attention again, but now between intermediate decoder representations H' and final encoder representations $H^{(\text{enc})}$:

$$Z = \underbrace{\text{softmax}\big(H'A\,B^{\top}H^{(\text{enc})^{\top}}\big)}_{\text{src-trg attention}}\;H^{(\text{enc})}C$$

$$H^{(\text{dec})} = \text{feed-forward}(\text{layer-norm}(H' + Z))$$

We predict target words with $H^{(\text{dec})}W_{out}$.

2.2 Features

In the spirit of minimalism, we follow the 80/20 principle (Pareto, 1896) and aim to achieve 80% of the translation quality with 20% of a common toolkit's code size. For this purpose we identified the most common features (the bare necessities) in recent works and implementations.[2] It includes standard architectures (see §2.1), label smoothing, dropout in multiple places, various attention mechanisms, input feeding, configurable encoder/decoder bridge, learning rate scheduling, weight tying, early stopping criteria, beam search decoding, an interactive translation mode, visualization/monitoring of learning progress and attention, checkpoint averaging, and more.

2.3 Documentation

The code itself is documented with doc-strings and in-line comments (especially for tensor shapes), and modules are tested with unit tests. The documentation website[3] contains installation instructions, a walk-through tutorial for training, tuning and testing an NMT model on a toy task[4], an overview of code modules, and a detailed API documentation. In addition, we provide thorough answers to frequently asked questions regarding usage, configuration, debugging, implementation details and code extensions, and recommend resources, such as data collections, PyTorch tutorials and NMT background material.

Counts	OpenNMT-py	XNMT	Joey NMT
Files	94	82	20
Code	10,287	11,628	2,250
Comments	3,372	4,039	1,393
Comment/Code Ratio	0.33	0.35	**0.62**

Table 1: Python code statistics for OpenNMT-py (commit hash `624a0b3a`), XNMT (`a87e7b94`) and Joey NMT (`e55b615`).

2.4 Code Complexity

In order to facilitate fast code comprehension and navigation (Wiedenbeck et al., 1999), Joey NMT objects have at most one level of inheritance. Table 1 compares Joey NMT with OpenNMT-py and XNMT (selected for their extensibility and thoroughness of documentation) in terms of code statistics, i.e. lines of Python code, lines of comments and number of files.[5] OpenNMT-py and XNMT have roughly 9-10x more lines of code, spread across 4-5x more files than Joey NMT . These toolkits cover more than the essential features for NMT (see §2.2), in particular for other generation or classification tasks like image captioning and language modeling. However, Joey NMT's comment-to-code ratio is almost twice as high, which we hope will give code readers better guidance in understanding and extending the code.

2.5 Benchmarks

Our goal is to achieve a performance that is comparable to other NMT toolkits, so that novices can start off with reliable benchmarks that are trusted by the community. This will allow them to build on Joey NMT for their research, should they want to do so. We expect novices to have limited resources available for training, i.e., not more than one GPU for a week, and therefore we focus on benchmarks that are within this scope. Pretrained models, data preparation scripts and configuration files for the following benchmarks will be made available on `https://github.com/joeynmt/joeynmt`.

[2] We refer the reader to the additional technical description in `https://arxiv.org/abs/1907.12484`: Table 6 in Appendix A.1 compares Joey NMT's features with several popular NMT toolkits and shows that Joey NMT covers all features that those toolkits have in common.

[3] `https://joeynmt.readthedocs.io`

[4] Demo video: `https://youtu.be/PzWRWSIwSYc`

[5] Using `https://github.com/AlDanial/cloc`

System	Groundhog RNN		Best RNN			Transformer	
	en-de	lv-en	layers	en-de	lv-en	en-de	lv-en
NeuralMonkey	13.7	10.5	1/1	13.7	10.5	–	–
OpenNMT-Py	18.7	10.0	4/4	22.0	13.6	–	–
Nematus	23.9	14.3	8/8	23.8	14.7	–	–
Sockeye	23.2	14.4	4/4	25.6	15.9	27.5	18.1
Marian	23.5	14.4	4/4	25.9	16.2	27.4	17.6
Tensor2Tensor	–	–	–	–	–	26.3	17.7
Joey NMT	23.5	14.6	4/4	26.0	15.8	27.4	18.0

Table 2: Results on WMT17 `newstest2017`. Comparative scores are from Hieber et al. (2018).

WMT17. We use the settings of Hieber et al. (2018), using the exact same data, pre-processing, and evaluation using WMT17-compatible SacreBLEU scores (Post, 2018).[6] We consider the setting where toolkits are used out-of-the-box to train a Groundhog-like model (1-layer LSTMs, MLP attention), the 'best found' setting where Hieber et al. train each model using the best settings that they could find, and the Transformer base setting.[7] Table 2 shows that Joey NMT performs very well compared against other shallow, deep and Transformer models, despite its simple code base.[8]

IWSLT14. This is a popular benchmark because of its relatively small size and therefore fast training time. We use the data, pre-processing, and word-based vocabulary of Wiseman and Rush (2016) and evaluate with SacreBLEU.[9] Table 3 shows that Joey NMT performs well here, with both its recurrent and its Transformer model. We also included BPE results for future reference.

System	de-en
Wiseman and Rush (2016)	22.5
Bahdanau et al. (2017)	27.6
Joey NMT (RNN, word)	27.1
Joey NMT (RNN, BPE32k)	27.3
Joey NMT (Transformer, BPE32k)	31.0

Table 3: IWSLT14 test results.

3 User Study

The target group for Joey NMT are novices who will use NMT in a seminar project, a thesis, or an internship. Common tasks are to re-implement a paper, extend standard models by a small novel element, or to apply them to a new task. In order to evaluate how well novices understand Joey NMT, we conducted a user study comparing the code comprehension of novices and experts.

3.1 Study Design

Participants. The novice group is formed of eight undergraduate students with a Computational Linguistics major that have all passed introductory courses to Python and Machine Learning, three of them also a course about Neural Networks. None of them had practical experience with training or implementing NMT models nor PyTorch, but two reported theoretic understanding of NMT. They attended a 20h crash course introducing NMT and Pytorch basics.[10] Note that we did not teach Joey NMT explicitly in class, but the students independently completed the Joey NMT tutorial.

As a control group (the "experts"), six graduate students with NMT as topic of their thesis or research project participated in the study. In contrast to the novices, this group of participants has a solid background in Deep Learning and NMT, had practical experience with NMT. All of them had previously worked with NMT in PyTorch.

Conditions. The participation in the study was voluntary and not graded. Participants were not allowed to work in groups and had a maximum

[6] `BLEU+case.mixed+lang.[en-lv|en-de]+numrefs.1+smooth.exp+test.wmt17+tok.13a+version.1.3.6`

[7] Note that the scores reported for other models reflect their state when evaluated in Hieber et al. (2018).

[8] Blog posts like Rush (2018) and Bastings (2018) also offer simple code, but they do not perform as well.

[9] `BLEU+case.lc+numrefs.1+smooth.exp+tok.none+version.1.3.6`

[10] See §?? in the supplemental material of `https://arxiv.org/abs/1907.12484` for details.

time of 3h to complete the quiz. They had previously locally installed Joey NMT[11] and could browse the code with the tools of their choice (IDE or text editor). They were instructed to explore the Joey NMT code with the help of the quiz, informed about the purpose of the study, and agreed to the use of their data in this study. Both groups of participants had to learn about Joey NMT in a self-guided manner, using the same tutorial, code, and documentation. The quiz was executed on the university's internal e-learning platform. Participants could jump between questions, review their answers before finally submitting all answers and could take breaks (without stopping the timer). Answers to the questions were published after all students had completed the test.

Question design. The questions are not designed to test the participant's prior knowledge on the topic, but to guide their exploration of the code. The questions are either free text, multiple choice or binary choice. There are three blocks of questions:[12]

1. **Usage of Joey NMT** : nine questions on how to interpret logs, check whether models were saved, interpret attention matrices, pre-/post-process, and to validate whether the model is doing what it is built for.

2. **Configuring Joey NMT** : four questions that make the users configure Joey NMT in such a way that it works for custom situations, e.g. with custom data, with a constant learning rate, or creating model of desired size.

3. **Joey NMT Code**: eighteen questions targeting the detailed understanding of the Joey NMT code: the ability to navigate between python modules, identify dependencies, and interpret what individual code lines are doing, hypothesize how specific lines in the code would have to get changed to change the behavior (e.g. working with a different optimizer). The questions in this block were designed in a way that in order to find the correct answers, every python module contained in Joey NMT had to be visited at least once.

Every question is awarded one point if answered correctly. Some questions require manual grading, most of them have one correct answer. We record overall completion time and time per question.[13]

3.2 Analysis

Total duration and score. Experts took on average 77 min to complete the quiz, novices 118 min, which is significantly slower (one-tailed t-test, $p < 0.05$). Experts achieved on average 82% of the total points, novices 66%. According to the t-test the difference in total scores between groups is significant at $p < 0.05$. An ANOVA reveals that there is a significant difference in total duration and scores within the novices group, but not within the experts group.

Per question analysis. No question was incorrectly answered by everyone. Three questions (#6, #11, #18) were correctly answered by everyone–they were appeared to be easiest to answer and did not require deep understanding of the code. In addition, seven questions (#1, #13, #15, #21, #22, #28, #29) were correctly answered by all experts, but not all novices–here their NMT experience was useful for working with hyperparameters and peculiarities like special tokens. However, for only one question, regarding the differences in data processing between training and validation (#16), the difference between average expert and novice score was significant (at $p < 0.05$). Six questions (#9, #18, #21, #25, #31) show a significantly longer average duration for novices than experts. These questions concerned post-processing, initialization, batching, end conditions for training termination and plotting, and required detailed code inspection.

LME. In order to analyze the dependence of scores and duration on particular questions and individual users, we performed a linear mixed effects (LME) analysis using the R library `lme4` (Bates et al., 2015). Participants and questions are treated as random effects (categorical), the level of expertise as fixed effect (binary). Duration and score per question are response variables.[14] For both response variables the variability is higher

[11] Joey NMT commit hash `0708d596`, prior to the Transformer implementation.

[12] https://arxiv.org/abs/1907.12484 contains the full list of questions, complete statistics and details of the LME analysis.

[13] Time measurement is noisy, since full minutes are measured and students might take breaks at various points in time.

[14] Modeling expertise with higher granularity instead of the binary classification into expertise groups (individual variables for experience with PyTorch, NMT and background in deep learning) did not have a significant effect on the model, since the number of participants is relatively low.

depending on the question than on the user (6x higher for score, 2x higher for time). The intercepts of the fixed effects show that novices score on average 0.14 points less while taking 2.47 min longer on each question than experts. The impact of the fixed effect is significant at $p < 0.05$.

3.3 Findings

First of all, we observe that the design of the questions was engaging enough for the students because all participants invested at least 1h to complete the quiz voluntarily. The experts also reported having gained new insights into the code through the quiz. We found that there are significant differences between both groups: Most prominently, the novices needed more time to answer each question, but still succeeded in answering the majority of questions correctly. There are larger variances within the group of novices, because they had to develop individual strategies to explore the code and use the available resources (documentation, code search, IDE), while experts could in many cases rely on prior knowledge.

4 Conclusion

We presented Joey NMT, a toolkit for sequence-to-sequence learning designed for NMT novices. It implements the most common NMT features and achieves performance comparable to more complex toolkits, while being minimalist in its design and code structure. In comparison to other toolkits, it is smaller in size and but more extensively documented. A user study on code accessibility confirmed that the code is comprehensibly written and structured. We hope that Joey NMT will ease the burden for novices to get started with NMT, and can serve as a basis for teaching.

Acknowledgments

We would like to thank Sariya Karimova, Philipp Wiesenbach, Michael Staniek and Tsz Kin Lam for their feedback on the early stages of the code and for their bug fixes. We also thank the student and expert participants of the user study.

References

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2017. An actor-critic algorithm for sequence prediction. In *ICLR*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Jasmijn Bastings. 2018. The annotated encoder-decoder with attention.

Douglas Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2015. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48.

Jindřich Helcl and Jindřich Libovický. 2017. Neural Monkey: An Open-source Tool for Sequence Learning. *PBML*.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2018. The sockeye neural machine translation toolkit at amta 2018. In *AMTA*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Vincent Nguyen, Jean Senellart, and Alexander Rush. 2018. Opennmt: Neural machine translation toolkit. In *AMTA*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Sachan, Philip Arthur, Pierre Godard, John Hewitt, Rachid Riad, and Liming Wang. 2018. XNMT: The extensible neural machine translation toolkit. In *AMTA*.

Vilfredo Pareto. 1896. *Cours d'économie politique: professé á l'Université de Lausanne*, volume 1. F. Rouge.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *WMT*.

Alexander Rush. 2018. The annotated transformer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Susan Wiedenbeck, Vennila Ramalingam, Suseela Sarasamma, and Cynthia L Corritore. 1999. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11(3):255–282.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*, Austin, Texas.